REMARKS

In reopening prosecution after applicants had appealed the final rejection, the Examiner has now rejected claims 1-5, 7-9 and 11-15 under 35 USC § 102(e) as being anticipated by Achenson et al. U.S. Patent No. 6,477,586. Claims 6, 10 and 16 stand rejected under 35 USC § 103(a) as being obvious from Achenson in view of Voll et al U.S. Patent No. 6,170,018. Applicants respectfully traverse these rejections and request reconsideration.

Applicants have amended claims 1, 7 and 11 to incorporate the subject matter of claims 6, 10 and 16, respectively, which specifies that the program structure destroys the first thread in the memory structure after the first thread completes a desired amount of work. Claims 5, 9 and 15 have been amended to recite that the first thread is reused to process other work before it is destroyed. No new matter has been added.

As amended, claims 1, 7 and 11 recite a method of parallel processing in a memory structure utilizing two threads which each represent an independent flow of control managed by separate program structures. A first thread is created in the memory structure which has two states: 1) processing work for the program structure, and 2) undispatched awaiting work to process. The method involves using the second thread to prepare work for the first thread to process and placing the work in a queue. If the first thread is awaiting work to process when the work is placed in the queue, the first thread is dispatched and processes the work in the queue. If the first thread is processing other work when the second thread place the work in the queue, the first thread completes

processing of the other work then accesses the work and processes it from the queue. After the first thread completes a desired amount of work, the program structure destroys it in the memory structure.

As recited in the specification at pages 19-21, this aspect of the invention allows the first thread to be treated as a data object by the software. The first thread may be created, reused as desired by the program structure, and destroyed after it completes a desired amount of work.

The Achenson patent discloses a distributed processing system in which messages are sent through different processes, from Process 1 to Process N. Column 4, lines 48-51. Each Process has its own local memory structure. Column 7, line 13. Within each Process there is a dispatcher thread that passes a RPC (remote procedure call) message to an appropriate available thread from a pool of workers within the Process to permit the message to be processed. Column 5, lines 55-59. If the worker thread in the Process is appropriate, it indicates that the RPC request is to be forwarded to another Process. Column 5, lines 60-63 and Column 6, line 64 to Column 7, line 2. Each process is connected to another Process (Column 5, lines 64-66) and the RPC request may be transferred to another Process. Column 7, lines 2-3.

Achenson neither creates nor destroys threads in each memory structure in each Process; they are simply always there, whether used or not, in a preexisting pool of workers. The Examiner has admitted that Achenson does not disclose the destruction of threads, but has taken the position that such step is obvious from Voll. Voll is only cited for its disclosure that a thread may be destroyed when processing by it completes.

However, if Voll's process of destroying threads upon completion of processing were implemented in the Achenson distributed processing system, then the Achenson system would be inoperative. This is because Achenson relies upon the worker threads in each Process to either perform the work requested, or pass the RPC request on to another Process. Achenson's disclosure underscores the criticality that worker threads not be destroyed when he states that "each of the said process compris[es] a plurality of threads" (Column 2, lines 17-18) and "each process must maintain a data structure which reflects which process is able to respond to a given RPC request." Column 8, lines 3-5. Since it is the worker threads in each Achenson Process that either performs the work or determines whether another process can perform the work, and each Process is connected to another by only a "single connection" (Column 5, lines 64-66), the absence by destruction of the worker threads after completion of work a particular process would bring Achenson's distributed processing system to a halt since the next request would have nowhere to go to be either completed or transferred.

Accordingly, one of ordinary skill in this art would not combine the disclosures of Achenson and Voll in the manner envisioned by the Examiner. The present invention as defined by applicants' claims is not prima facie obvious since the hypothetical combination of elements is chosen solely as a result of the hindsight benefit of reading applicants' own specification.

Applicants' invention creates threads on demand to perform work from a queue, and then destroys those threads after a desired amount of work is performed. It is not relegated to a fixed distribution structure, as Achenson is, but instead permits direct

manipulation of data by allowing a thread (or flow of control) to be treated as a data object by the software, and created, used, reused and destroyed as needed. See, page 1, lines 20-28.

For the reasons given above, applicants submit that the claims of the instant application are in condition for allowance. Reconsideration of the rejection and allowance of the claims re respectfully requested. Any questions which may be handled by telephone should be directed to the undersigned at (203) 787-0595.

Respectfully submitted,

Peter W. Peterson
Reg. No. 31,867

DeLIO & PETERSON, LLC
121 Whitney Avenue
New Haven, CT 06510-1241
(203) 787-0595

ibmf100275000amdB.doc